مخطوطتك الأولى بالشيل

إعداد: فيصل شامخ

chamfay@gmail.com

بسم الله الرحمن الرحيم

مقدمة:

نظرا لأن سطر الأوامر مهم جدا في لينكس حتى بوجود الواجهة الرسومية، حيث أن هناك من المهام التي تعجز الواجهة الرسومية عن أدائها، ولأنك تستطيع أن تفعل بنظامك ما تريد من خلال سطر الأوامر، وأنه يمكنك كذلك كتابة ما تريد من أوامر في ملف وتقوم بتنفيذه دفعة واحدة، أردت أن أُعدّ هذا الكتاب القصير مبينا فيها شيئا ولو يسيرا للمبتدئين في كتابة أوامر الشيل وهذا لأنني مبرمج شيل سكربت مبتدأ.

فائدة المخطوطة (script) كما قلت تنفيذ مجموعة من الأوامر دفعة واحدة وكذلك في حالة عدة أوامر طويلة ومعقدة بحيث تصبح كتابتها مملة وبالتالي يجب كتابتها في ملف ليتسنى لنا مراجعتها، زد على ذلك إعادة استعمالها مرة أخرى ونقلها لمن تريد.

الشيل shell مثبة مسبقا في أي نظام لينكس فهي تلعب دور المترجم بينك وبين النواة وبالتالي ضمان محمولية السكربت. خاصة أن bash هي الافتراضية معظم توزيعات لينكس إن لم نقل كلها.

يحتوي الكتاب على المفاهيم الأساسية لكتابة سكربت (المتغيرات،الشروط، الحلقات، الدوال، القوائم...) مع شرح أرى أنه سهل ومختصر مع بعض الأمثلة إن أمكن وتبقى زيادة التعلم بالكتابة والتعود على الأوامر والبحث.

أرجو من الله أن ينفع بهذا الكتاب من يقرأه.

يوم:19 مارس 2011

سكربت مرحبا بالعالم:

نفتح أي محرر نصوص (gedit) ونكتب مايلي:

```
#!/bin/bash
echo "Hello World"
```

ثم نقوم بحفظه في ملف نسمية مثلا: your_script

السكربت به سطرين، الأول يعلم النظام بأننا نستخدم البرنامج bash في السكربت وهذا يجب كتابته دائما في بداية كل سكربت، والثاني أمر يقوم بطباعة العبارة Hello World على الشاشة (standard ouptout)

لتشغيل السكربت يجب إعطاؤه صلاحية التنفيذ:

```
chmod +x your_script
```

ومن ثم نقوم بتنفيذه بكتابة:

```
./your_script
```

سكربت آخر:

```
#!/bin/bash
echo "Your home path is: $HOME"
echo "Current Directory is: $PWD"
```

السطر الثاني يطبع مسار مجلد المنزل والثالث يطبع المسار الحالي الذي به ملف السكربت.

التعليقات:

التعليقات هي عبارة عن نصوص نكتبها داخل السكربت لتوضح عمل بعض أجزاء السكربت وهي لا تنفذ من طرف bash لكننا نكتبها لنسهل إعادة فهم الأجزاء التي قمنا بكتابتها وخاصة في حالة السكربت الطويل والمعقد.

لكتابة تعليق نسبقة بالعلامة # ثم التعليق الذي نريد.

```
#!/bin/bash
x=10       # assign 10 to variable x
```

المتغيرات:

المتغير (كما في لغات البرمجة) هو مكان في الذاكرة يحتوي على قيمة معينة، في الشيل يمكن للمتغير أن يكون عدد، حرف أو سلسلة نصية ولا حاجة للتصريح عن متغير أو نوعه يكفي أن نكتب اسمه متبوعا بـ = ثم قيمته.

مثال:

```
#!/bin/bash
my_str="Hello World!"
echo $my_str         # print Hello World!
echo my_str          # print my_str
```

هنا صرحنا عن متغير my_str وأسندنا له قيمة وهي: Hello World! (يجب عدم ترك مسافة قبل أو بعد علامة المساواة).

للحصول على قيمة هذا المتغير نضع $ قبل اسم المتغير، نفذ السكربت السابق وسترى الفرق بين السطرين الثالث والرابع.

هناك المتغيرات المعرفة مسبقا من طرف النظام وتكون مكتوبة بأحرف كبيرة مثلا:

PWD : قيمتة سلسلة نصية تحتوي على المسار الحالي.

HOME : قيمته سلسلة نصية تمثل مسار مجلد المنزل.

PATH : يحتوي على المسارات التي يبحث فيها النظام لتنفيذ أمر.

BASH : يحتوي على مسار bash......

العوامل الشرطية:

نحتاج في بعض الحالات أن نفحص قيمة متغير أو مخرج أمر معين لنقرر من خلاله تنفيذ أوامر أخرى أو عدم تنفيذها بناء على نتيجة الفحص

التركيب if..then:

```
if [ condition ]
then
      expression
fi
```

يعني إذا تحقق الشرط condition ننفذ إذا التعليمة أو التعليمات expression.

fi تعني نهاية الفحص (if مقلوبة)

```
#!/bin/bash
name="fayssal"
if [ "$name" = "fayssal" ]
then
      echo "Your name: $name"    # print fayssal
fi
```

التركيب if..then..else

```
if [ condition ]
then
      expression1
else
      expression2
fi
```

يعني إذا تحقق الشرط condition نُنفذ إذا التعليمة expression1 وإذا لم يتحقق ننفذ التعليمة expression2.

مثال:

```
#!/bin/bash
read -p "Enter your name: " name
if [ "$name" = "fayssal" ] ; then
      echo "You are $name"
else
      echo "Your aren't fayssal"
fi
```

الأمر read يقوم بقراءة المعطيات التي يدخلها المستخدم ومن ثم تخزينها في المتغير name.

يمكن كتابة عدة أوامر في سطر واحد وذلك بفصلها بالعلامة ; كما بالمثال أعلاه.

التركيب if..then..elif..else:

```
if [ condition1 ]; then
      expression1
elif [ condition2 ]; then
      expression2
```

```
else
      expression3
fi
```

في هذه الحالة يقوم التركيب بفحص شرطين إذا كان condition1 صحيح ننفذ expression1 ثم نفحص الشرط الثاني وفي حال عدم تحقق أي من الشرطين ننفذ ما بعد else.

عمليات المقارنة:

المقارنة بين السلاسل النصية:

استعملنا فيما سبق المعامل = للمقارنة، فيما يلي جدول يوضح المعاملات الأخرى.

العملية	وظيفتها
`$str1 = $str2 or ($str1 == $str2)`	هل السلسلة str1 تساوي السلسلة str2
`$str1 != $str2`	هل السلسلة str1 لا تساوي السلسلة str2
`-z $str`	هل السلسلة str فارغة
`-n $str`	هل السلسلة srt غير فارغة

المقارنة بين الأعداد:

العملية	وظيفتها
`$number1 -eq $number2`	هل الرقم number1 يساوي الرقم number2
`$number1 -ne $number2`	هل الرقم number1 لا يساوي الرقم number2
`$number1 -gt $number2`	هل الرقم number1 أكبر من الرقم number2
`$number1 -ge $number2`	هل الرقم number1 أكبر أو يساوي الرقم number2
`$number1 -lt $number2`	هل الرقم number1 أقل من الرقم number2
`$number1 -le $number2`	هل الرقم number1 أقل أو يساوي الرقم number2

عمليات على الملفات:

العملية	وظيفتها
`-e ~/file.txt`	فحص هل الملف file.txt موجود.
`-d /var`	فحص هل var عبارة عن مجلد.
`-f /bin/bash`	فحص هل bash عبارة عن ملف.

-L /bin/sh	فحص هل sh عبارة هن اختصار (shortcut)
-r ~/file.txt	فحص هل الملف file.txt قابل للقراءة.
-w ~/file.txt	فحص هل الملف file.txt قابل للكتابة عليه.
-x /bin/bash	فحص هل الملف bash ملف تنفيذي.

التركيب case..in:

```
case choice in
first)
	1st expression
	;;
second)
	2nd expression
	;;
*)
	3rd expression
	;;
esac
```

لتجنب تكرار elif عدة مرات في حالة فحص عدة شروط نستخدم التركيب case..in

ويعني إذا كانت قيمة choise مساوية ل first ننفذ 1st expression وإذا كانت تساوي second ننفذ 2nd expression وإلا ننفذ الخيار * (مثل else في if)، المعامل ;; يسمح بالخروج من التركيب بعد تنفيذ ماقبله من أوامر.

يجب عدم نسيان الكلمة esac (مقلوب case) لتفيد نهاية التركريب case..in

```
#!/bin/bash

name="fayssal"
case $name in
"fayssal")		# if $name = fayssal
	echo "$name"
	;;
"tarek")		# if $name = tarek
	echo "$name"
	;;
*)			# else
	echo "anonym"
	;;
esac
```

الحلقات loops:

نستعمل الحلقات عندما نحتاج إلى تكرار أمر/أوامر عدة مرات محددة أو إلى أن يتحقق شرط معين.

الحلقة for:

```
#!/bin/bash
for nbr in 1 2 3 4 5
do
	echo "Number: $nbr"
done
```

هنا تبدأ الحلقة بأن يأخذ المتغير nbr الرقم الأول من السلسلة ثم الثاني ... إلى آخر السلسلة وعندها يتم الخروج من الحلقة وفي كل حالة يتم تنفيذ الأوامر ما بين do و done، والناتج يكون كالتالي:

```
fayssal@fayssal-desktop:~/Desktop$ ./shell.sh
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

يمكن استخدام الحلقة كما يلي:

```
#!/bin/bash
for i in $( ls $HOME ); do
      echo $i
done
```

والناتج يكون:

```
fayssal@fayssal-desktop:~/Desktop$ ./shell.sh
Desktop
Documents
Downloads
file.txt
Music
Pictures
Public
Templates
Videos
```

هنا الأمر `ls $HOME` يقوم بعرض الملفات والمجلدات الموجودة في مجلد المنزل ونتيجة هذا الأمر مررناها كمتغير للحلقة.

يمكن استخدام الأمر seq لتوليد سلسلة من الأرقام كما يلي:

```
#!/bin/bash
for i in $(seq 0 8) ; do
      echo $i
done
```

بحيث يأخذ i القيم من 0 إلى 8.

الحلقة while:

```
while condition
do
      instruction
done
```

معناها مادام الشرط condition صحيح نفذ الأمر insruction، ويتم الخروج من الحلقة في حالة عدم تحققه فقط.

```
#!/bin/bash
while [ "$name" != "ubuntu" ]; do
      read -p "Enter your name: " name
done
echo "Hello: $name"
```

هنا لن تنتهي الحلقة إلا إذا كتبت الكلمة ubuntu.

الحلقة until:

وهي عكس while ومعناها إلى أن يتحقق الشرط condition نفذ الأمر insruction، ويتم الخروج من الحلقة في

حالة تحققه فقط.

```
until condition ; do
      instruction
done
```

الدوال functions:

الدوال جزء من برنامج (مجموعة من الأوامر) تندرج تحت اسم واحد وتنفذ فقط عند استدعائها، فائدتها تجنب تكرار نفس الكود وتسهيل قراءة البرنامج وصيانته وهي موجودة في كل لغات البرمجة.

في bash يمكن كتابتها بإحدى التعبيرين:

```
funcion function_name {
   commands
}
```

أو:

```
function_name() {
   commands
}
```

حيث function_name هو اسم الدالة و commands هي مجموعة الأوامر.

مثال:

```
#!/bin/bash
function greeting {
     echo "How are you today?"
}
greeting              # call the function.
```

أولا قمنا بتعريف الدالة greeting ثم قمنا باستدعائها وذلك بكتابة اسمها فقط. والناتج:

```
fayssal@fayssal-desktop:~/Desktop$ ./shell.sh
How are you today?
```

استعمال الوسائط في الدوال:

استعمال الوسائط parameters مهم في الدالة لجعلها أوسع في الاستعمال، فمثلا لتعريف دالة بسيطة تقوم بإدخال اسم كوسيط ومن ثم تقوم بطباعته تحية للمستخدم نكتب مثلا:

```
#!/bin/bash
function greeting {
      echo "Hello: $1"
}
greeting Fayssal
```

الرمز 1$ الوسيط الأول فالدالة لديها وسيط واحد، أنظر المثال التالي:

```
#!/bin/bash
function calculat {
case $2 in
"+")                    # Plus operation.
      let "y=$1$2$3"
      echo $y
      ;;
"-")                    # Minus operation.
      let "y=$1$2$3"
      echo $y
      ;;
```

```
*)
      echo "Invalid operation!"
      ;;
esac
}
calculat 10 + 5
calculat 10 – 5
calculat 10 ++ 10
```

الدالة calculat تستقبل 3 وسائط الأول والثالث هما طرفا العملية الحسابية و الثاني هو نوع العملية.

هنا استدعينا الدالة مرتين، عند التنفيذ نرى النتيجة!

```
fayssal@fayssal-desktop:~/Desktop$ ./shell.sh
15
5
Invalid operation!
```

القوائم select:

إذا أردت عرض قائمة في سطر الأوامر تعرض فيها على المستخدم عدة خيارت نستعمل الكلمة المفتاحية select.

```
select choise in sequence
do
      commands...
done
```

حيث choise متغير يأخذ في كل مرة قيمة من squence وتقوم بطباعته وهي تشبه إلى حد ما الحلقة إلا أنها في الأخير تطلب من المستخدم إدخال رقم أحد الخيارات المطبوعة على الشاشة نأخذ مثال ليتضح الأمر أكثر.

```
#!/bin/bash
select choise in Enter Quit
do
      if [ $choise = "Enter" ]; then
            echo Hello!
      elif [ $choise = "Quit" ]; then
            echo "Goodbye!"
            exit
      else
            clear;
            echo "Please select an option."
      fi
done
```

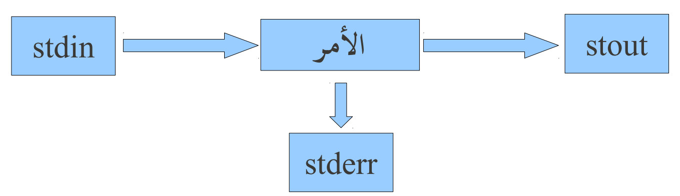
يكون الناتج على الشاشة ما يلي:

```
fayssal@fayssal-desktop:~/Desktop$ ./shell.sh
1) Enter
2) Quit
#?
```

كل ما عليك فعله هو كتابة رقم الخيار والضغط على Enter في لوحة المفاتيح.

إعادة التوجيه redirection:

أولا يوجد ثلاث وجهات لأي **أمر** في الشيل الأولى stdin حيث يقوم الأمر من خلاله باستقبال المدخلات، و stdout يمكن للأمر أن يقوم بطبع مخرجاته والثالث strerr في حالة خطأ في تنفيذ هذا الأمر يطبع مخرجاته في stderr.



يمكننا إعادة توجيه هذه المعطيات من سطر الأوامر إلى ملف أو قراءتها من ملف

إعادة توجيه stdout إلى ملف:

```
#!/bin/bash
echo "You home path is: $HOME" > file.txt
```

سيتم كتابة مخرجات الأمر ls إلى الملف file.txt عوضا عن كتابتها على سطر الأوامر، فعند فتحه نجد التالي:

```
fayssal@fayssal-desktop:~$ cat file.txt
Your home path is: /home/fayssal
```

في هذا المثال، في حالة وجود الملف السابق سيتم فقدان محتواه السابق، أم إذا أردت الكتابة في نهاية الملف نستعمل المعامل >> كما بالمثال:

```
#!/bin/bash
echo "Current path is: $PWD" >> file.txt
```

فنحصل على التالي:

```
fayssal@fayssal-desktop:~$ cat file.txt
Your home path is: /home/fayssal
Current Dicrectory is: /home/fayssal/Desktop
```

توجيه stderr إلى ملف:

```
#!/bin/bash
ls File 2> log.txt
```

إذا الملف File غير موجود سيعطي الأمر رسالة خطأ وبدلا من طبعها على الشاشة تُوجه إلى الملف log.txt ويكون محتواه مايلي:

```
fayssal@fayssal-desktop:~$ cat log.txt
ls: cannot access File: No such file or directory
```

توجيه stdout و stderr معا إلى ملف:

لتوجيه أي مخرجات لأمر معين إلى ملف نكتب:

```
#!/bin/bash
ls File &> log.txt
```

وهذا مفيد مثلا في حالة أردنا أن يتم الأمر في صمت بدون أي مخرجات ويمكن توجيهه إلى الملف null.

```
#!/bin/bash
ls File &> log.txt > /dev/null
```

الملف null يسمى ملف العدم يعني أي معطيات نقوم بكتابتها فيه تعدم ويكون محتواه لاشيء.

الأنابيب Pipes:

المعامل الأنبوبي | وظيفته سهلة ربط stdin لأمر معين إلى stdout لأمر آخر، و بكل بساطة ربط مخرج أمر إلى مدخل أمر آخر.

مثال:

```
#!/bin/bash
ls | grep "D"
```

هنا مخرج الأمر ls (يعرض جميع الملفات والمجلدات) ربطناه إلى مدخل الأمر grep (يبحث عن السطر التي بها الحرف D) والناتج تتم كتابته على الشاشة.

للعلم يمكن استخدام سلسلة من هذا المعاملات.

```
cmd1 | cmd2 | cmd3
```

تنفيذ العمليات الحسابية:

نفذ المثال التالي وقارن:

```
#!/bin/bash
echo 10 + 15           # print 10 + 15
echo $((10 + 15))      # print 25
echo $[10 + 15]        # print 25
```

في السطر الثاني echo تكتب 10+15 كما هي، لكن في السطرين الثالث والرابع ناتج العملية الحسابية.

```
#!/bin/bash
let "y = 10 + 10"
echo $y         # print 20
```

تنفيذ عدة أوامر في سطر واحد:

لتنفيذ عدة أوامر في نفس السطر نفصل بينهم بفاصلة منقوطة ;

```
#!/bin/bash
for a in 1 2 3 4 5; do echo "a equal: $a"; done
```

وضع عدة أوامر في كتلة واحدة:

نستعمل الحاضنتين {} أو القوسين () لكن هناك اختلاف بينهما.

```
#!/bin/bash
variable1="ubuntu"
{
      variable1="World!";
}
variable2="ubuntu"
(
      variable2="World!";
)
echo Hello $variable1      # print Hello World!
echo Hello $variable2      # print Hello ubuntu
```

نلاحظ حالتين:

1- تغيرت قيمة المتغير variable1 من ubuntu إلى World!.

2- في المقابل قيمة variable2 لم تتغير (بقيت ubuntu).

في الحقيقة، المتغير variable2 الذي بين القوسين () ليس نفسه الذي خارجهما والذي أظهرته الدالة echo. فالأول معرف في برنامج جزئي آخر بدايته "(" ونهايته ")".

المعاملات في السكربت:

كما رأينا بالنسبة للدوال نستخدم العلامة $ متبوعة برقم كذلك بالنسبة لملف السكربت.

أنشئ ملفا اسمه مثلا shell.sh واكتب فية ما يلي:

```
#!/bin/bash
echo "Filename: $0"
echo "Parameter1: $1"
echo "Parameter1: $2"
```

ثم اكتب في الطرفية (يجب أن تكون مسار الملف shell.sh) ما يلي:

```
fayssal@fayssal-desktop:~/Desktop$ chmod +x shell.sh
fayssal@fayssal-desktop:~/Desktop$ ./shell.sh One Two
Filename: ./shell.sh
Parameter1: One
Parameter1: Two
```

نلاحظ جليا أن الرمز 0$ يحتوي على اسم الملف، الرمز 1$ يحتوي على المعامل الأول الذي قمنا بإدخاله (One) وهكذا..

ملاحظة : يمكن استعمال 9 معاملات.

المراجع:

- BASH Programming – Introduction HOW–TO
- موقع siteduzero على هذا الرابط: http://www.siteduzero.com/tutoriel-3-12827-reprenez-le-controle-a-l-aide-de-linux.html#part_88347
- كتاب: دليل المستخدم العربي فى أوامر لينكس .